A. Experimental Results

Video Regression, Interpolation and Inpainting. For video regression, we provide the complete PSNR and MS-SSIM results on the Buuny and UVG datasets in Table 10 and Table 11. Besides, MS-SSIM results for video interpolation and inpainting on the UVG and DAVIS validation datasets are reported in Table 12 and Table 16, respectively. Further visualizations are available in Fig. 9 and Fig. 10.

Encoding Time. The encoding time comparison for video regression is shown in Table 13. Our enhanced models exhibit about a twofold increase in encoding latency.

Additional baselines. Apart from the classic NeRV, E-NeRV and HNeRV, our boosted method can also apply to recently proposed implicit video models. Table 14 reports the regression results of DNeRV [15], DivNeRV [55] and their boosted versions on the UVG dataset. For DNeRV, we follow its setting to crop videos into 1024×1920. The results show that our approach boosts DNeRV and DivNeRV by 0.77dB and 1.42dB PSNR, respectively.

Ablation Study. To further verify the contribution of each boosting component, a set of ablation experiments are conducted in Table 15. Since NeRV, E-NeRV, and HNeRV have an equal number of parameters, they shares similar upper bounds in model fitting capabilities. Compared with original INR baselines, the integration of temporal-aware affine transform (TAT) allows intermediate features to effectively align with the target frame, enhancing the reconstruction quality. Subsequently, we introduce a sinusoidal NeRV-like (SNeRV) block, facilitating diverse feature generation and balanced parameter distribution. Unlike NeRV and E-NeRV, which use an index for frame generation, HNeRV introduces content-aware embedding as prior information for each frame, facilitating efficient video fitting without elaborate feature extraction. Without the help of prior information, the SNeRV block significantly enhances NeRV and E-NeRV's performance by 2.78dB and 1.33dB, respectively. We further incorporate a high-frequency information-preserving loss that enables INRs to capture intricate video details more effectively.

B. Video Compression Details

To build video codecs, we fine-tune INR models using our consistent entropy minimization (CEM) method. For the entropy regularization term, we set κ as 0.05, 0.2, and 0.5 in the NeRV/E-NeRV/HNeRV, NeRV-Boost/E-NeRV-Bosst, and HNeRV-Boost, respectively. We use asymmetric numeral systems of the constriction library [4] to perform entropy coding. In recent learning-based video codecs, we obtain the results of DCVC [22] and DCVC-TCM [39] by employing their open source codes[1] with a Group of Pictures (GOP) size of 32. The I frame codecs in DCVC utilize the pretrained models *cheng-2020* [10] with quality indices 3,4,5,6 as provided by CompressAI [5]. For traditional codecs, we implement H.264 and H.265 with the veryslow preset and enabling B frames by using the following commands:

- H.264: *ffmpeg -pix_fmt yuv420p -s:v W×H -i Video.yuv -vframes* N_e *-c:v libx264 -preset veryslow -qp* QP *-g* GOP *output.mkv*
- H.265: *ffmpeg -pix_fmt yuv420p -s:v W×H -i Video.yuv -vframes* N_e *-c:v libx265 -preset veryslow -x265-params "qp=*QP*:keyint=*GOP*" output.mkv*

where N_e denotes the number of encoded frames and QP represents the number of quantization parameters. The GOP size is set as 32.

Bitrate computation. For INR-based codecs, we model the probability $p(\hat{\boldsymbol{y}}_t)$ of the quantized embedding as a Gaussian distribution. Then the estimated bitrate $R(\hat{\boldsymbol{y}}_t)$ of the quantized embedding can be expressed as:

$$R(\hat{\boldsymbol{y}}_t) = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}[-\log_2 p(\hat{\boldsymbol{y}}_t)] \tag{9}$$

The estimated bitrates $R(\hat{\boldsymbol{\theta}})$ and $R(\hat{\psi})$ follow the above similar computation procedure. In addition, when optimizing the RD trade-off, we need to choose the suitable B_{avg}. Fig. 8 illustrates the rate-distortion curves for various B_{avg} values on our boosted HNeRV. We see that $B_{avg} = 4$ or 5 bits yield superior RD performance compared to other scenarios. Based on empirical evaluation, B_{avg} is fixed as 4 bits.

C. Network Structure

Fig. 11 and Fig. 12 illustrate our proposed NeRV-Boost and E-NeRV-Boost frameworks. Given NeRV and E-NeRV baselines, our approach involves three key steps. Firstly, inserting TAT residual blocks after NeRV-like blocks. Secondly, replacing the GELU layer with a SINE layer in NeRV-like blocks. Finally, in the last three stages, stacking two sinusoidal NeRV-like blocks for feature upsampling and refinement. Since NeRV and E-NeRV utilize 3×3 kernels in their upsampling blocks, there is no need to alter the kernel size in the last three stages. These three revision steps effectively transform most video INR models into our INR-Boost models.

Table 17 and Table 18 give the network details of our condition decoders in different INR-Boost models. Our boosted models maintain the same channel reduction factor as their baseline counterparts. Model size variations are achieved by adjusting the channel width C_1. For the INR baselines, we employed their open-source codes, following specific training commands.

- NeRV[2]: *python train_nerv.py -e 300 --lower-width 12*

[1] https://github.com/microsoft/DCVC/tree/main

[2] https://github.com/haochen-rye/NeRV/tree/master

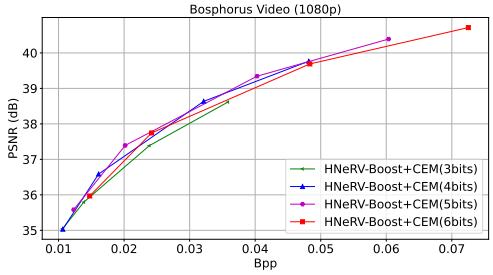

Figure 8. Comparisons between different B_{avg} bits.

--num-blocks 1 --dataset {dataset} --frame_gap 1 --outf {outf} --embed 1.25_80 --stem_dim_num 256_1 --reduction 2 --fc_hw_dim 9_16_{Depth} --expansion 1 --single_res --loss Fusion6 --warmup 0.2 --lr_type cosine --strides 5 3 2 2 2 --conv_type conv -b 1 --lr 0.0005 --norm none --act gelu

- E-NeRV[3]: *python main.py --output_dir {outf} --cfg_path [stem_dim_num: 256, fc_hw_dim: 9_16_Depth, stride_list: 5 3 2 2 2, lower_width:12, block_dim: 128, mlp_dim: 64]*
- HNeRV[4]: *python train_nerv_all.py --outf {outf} --data_path {dataset} –vid {dataset_name} --conv_type convnext pshuffel --act gelu --norm none –crop_list 1080_1920 --resize_list -1 --loss L2 --enc_strds 5 3 2 2 2 --enc_dim 64_16 --dec_strds 5 3 2 2 2 --ks 0_1_5 --reduce 1.2 --modelsize {Size} -e 300 --lower_width 12 -b 1 --lr 0.001*

[3] https://github.com/kyleleey/E-NeRV/tree/main
[4] https://github.com/haochen-rye/HNeRV

Table 10. Video regression results on the Bunny video in PSNR and MS-SSIM. The total size of HNeRV and HNeRV-Boost include the embedding size and the network size.

Model	Size	PSNR	PSNR Avg.	MS-SSIM	MS-SSIM Avg.
NeRV [7]	0.76M/1.49M/3.03M	26.12/28.54/31.84	28.83	0.8551/0.9183/0.9633	0.9122
NeRV-Boost	0.76M/1.51M/3.00M	**30.25/33.71/37.25**	**33.73**	**0.9341/0.9690/0.9860**	**0.9630**
E-NeRV [23]	0.76M/1.51M/3.00M	29.39/33.44/37.32	33.38	0.9404/0.9770/0.9899	0.9691
E-NeRV-Boost	0.75M/1.54M/3.07M	**32.61/37.19/40.07**	**36.62**	**0.9634/0.9857/0.9924**	**0.9805**
HNeRV [9]	0.75M/1.52M/3.03M	30.55/35.13/38.15	34.61	0.9229/0.9731/0.9874	0.9611
HNeRV-Boost	0.76M/1.51M/3.08M	**35.09/38.52/41.09**	**38.23**	**0.9749/0.9882/0.9934**	**0.9855**

Table 11. Video regression results on the UVG dataset in PSNR and MS-SSIM.

Model	Size	PSNR								MS-SSIM							
		Beauty	Bosph.	Honey.	Jockey	Ready.	Shake.	Yacht.	Avg.	Beauty	Bosph.	Honey.	Jockey	Ready.	Shake.	Yacht.	Avg.
NeRV [7]	3.04M	33.14	32.74	37.18	30.99	23.97	33.06	26.72	31.11	0.8918	0.9358	0.9806	0.8989	0.8426	0.9347	0.8712	0.9079
NeRV-Boost	3.06M	**33.55**	**34.51**	**39.04**	**32.82**	**26.08**	**34.54**	**28.76**	**32.76**	**0.8967**	**0.9480**	**0.9840**	**0.9174**	**0.8768**	**0.9458**	**0.8931**	**0.9231**
E-NeRV [23]	3.01M	33.29	33.87	38.88	28.73	23.98	34.45	27.38	31.51	0.8933	0.9444	0.9843	0.8708	0.8449	0.9468	0.8842	0.9098
E-NeRV-Boost	3.03M	**33.75**	**35.62**	**39.61**	**32.39**	**27.75**	**35.48**	**29.23**	**33.40**	**0.8987**	**0.9577**	**0.9854**	**0.9101**	**0.9057**	**0.9543**	**0.9015**	**0.9305**
HNeRV [9]	3.05M	33.36	33.62	39.17	32.31	25.60	34.90	28.33	32.47	0.8907	0.9320	0.9843	0.8948	0.8490	0.9479	0.8642	0.9090
HNeRV-Boost	3.05M	**33.80**	**36.11**	**39.65**	**34.28**	**28.19**	**35.88**	**29.33**	**33.89**	**0.8996**	**0.9653**	**0.9854**	**0.9298**	**0.9139**	**0.958**	**0.9019**	**0.9363**
NeRV [7]	5.07M	33.62	34.32	38.32	32.86	25.67	34.24	28.06	32.44	0.8994	0.9528	0.9832	0.9230	0.8854	0.9488	0.9015	0.9277
NeRV-Boost	5.00M	**33.89**	**35.86**	**39.31**	**34.16**	**27.78**	**35.33**	**30.00**	**33.76**	**0.9020**	**0.9608**	**0.9846**	**0.9332**	**0.9072**	**0.9564**	**0.9160**	**0.9372**
E-NeRV [23]	5.09M	33.77	35.38	39.33	31.56	25.37	35.23	28.64	32.76	0.9002	0.9596	0.9851	0.9050	0.8804	0.9561	0.9098	0.9280
E-NeRV-Boost	5.01M	**34.02**	**36.79**	**39.71**	**33.90**	**29.29**	**36.20**	**30.24**	**34.31**	**0.9026**	**0.9669**	**0.9856**	**0.9287**	**0.9283**	**0.9626**	**0.9181**	**0.9418**
HNeRV [9]	5.06M	33.84	34.49	39.56	33.64	27.24	35.73	29.29	33.40	0.8987	0.9430	0.9853	0.9114	0.8848	0.9588	0.8857	0.9240
HNeRV-Boost	5.01M	**34.14**	**37.87**	**39.74**	**35.84**	**30.36**	**36.71**	**30.77**	**35.06**	**0.9045**	**0.9764**	**0.9857**	**0.9467**	**0.9413**	**0.9675**	**0.9249**	**0.9496**
NeRV [7]	10.10M	34.10	36.52	39.35	35.37	28.10	35.82	30.11	34.20	**0.9088**	0.9701	0.9852	0.9493	0.9302	0.9662	0.9354	0.9493
NeRV-Boost	10.08M	**34.17**	**37.77**	**39.65**	**36.23**	**30.25**	**36.81**	**32.06**	**35.28**	0.9074	**0.9749**	**0.9855**	**0.9525**	**0.9419**	**0.9703**	**0.9429**	**0.9536**
E-NeRV [23]	10.16M	34.18	37.31	39.70	34.62	28.27	36.50	30.36	34.42	0.9065	0.9733	0.9858	0.9396	0.9297	0.9689	0.9361	0.9486
E-NeRV-Boost	10.04M	**34.28**	**38.39**	**39.82**	**35.88**	**31.42**	**37.34**	**31.94**	**35.58**	**0.9065**	**0.9767**	**0.9859**	**0.9481**	**0.9515**	**0.9730**	**0.9389**	**0.9544**
HNeRV [9]	10.07M	34.22	37.27	39.73	34.59	29.59	36.82	30.70	34.70	0.9053	0.9695	0.9857	0.9215	0.9255	0.9696	0.9134	0.9415
HNeRV-Boost	10.03M	**34.42**	**39.75**	**39.83**	**37.57**	**33.12**	**37.85**	**32.90**	**36.49**	**0.9096**	**0.984**	**0.9859**	**0.9617**	**0.9647**	**0.9768**	**0.9475**	**0.9615**
NeRV [7]	15.09M	34.36	37.66	39.59	36.55	29.81	36.86	31.43	35.18	**0.9170**	0.9766	**0.9857**	0.9588	**0.9509**	0.9741	0.9508	0.9591
NeRV-Boost	15.04M	**34.47**	**38.87**	**39.67**	**37.35**	**30.87**	**37.37**	**33.00**	**35.94**	0.9157	**0.9803**	0.9855	**0.9622**	0.9481	**0.9742**	**0.9527**	**0.9598**
E-NeRV [23]	15.02M	34.34	38.41	39.78	35.98	29.90	37.32	31.52	35.32	**0.9111**	0.9790	0.9860	0.9528	0.9492	0.9754	0.9491	0.9575
E-NeRV-Boost	15.06M	**34.40**	**39.31**	**39.85**	**36.90**	**32.46**	**37.99**	**33.00**	**36.27**	0.9089	**0.9819**	**0.9860**	**0.9574**	**0.9598**	**0.9776**	**0.9496**	**0.9602**
HNeRV [9]	15.02M	34.37	38.40	39.81	35.76	31.02	37.00	31.82	35.45	0.9079	0.9766	0.9859	0.9370	0.9435	0.9705	0.9295	0.9501
HNeRV-Boost	15.04M	**34.65**	**40.72**	**39.88**	**38.41**	**34.72**	**38.47**	**34.16**	**37.29**	**0.9176**	**0.9870**	**0.9861**	**0.9678**	**0.9739**	**0.9803**	**0.9578**	**0.9672**

Table 12. Video interpolation results on the UVG dataset in MS-SSIM.

Model	Beauty	Bosph.	Honey.	Jockey	Ready.	Shake.	Yacht.	Avg.
NeRV [7]	**0.8696**	0.9297	0.9797	**0.7542**	**0.7070**	0.9238	0.8578	0.8603
NeRV-Boost	0.8673	**0.9461**	**0.9835**	0.7357	0.6970	**0.9250**	**0.8708**	**0.8608**
E-NeRV [23]	0.8702	0.9383	0.9838	**0.7536**	0.7029	**0.9288**	0.8720	0.8642
E-NeRV-Boost	**0.8719**	**0.9525**	**0.9846**	0.7476	**0.7233**	0.9272	**0.8857**	**0.8704**
HNeRV [9]	0.8702	0.9379	0.9841	0.7677	0.7056	0.9263	0.8287	0.8601
HNeRV-Boost	**0.8754**	**0.9664**	**0.9849**	**0.7836**	**0.7527**	**0.9284**	**0.8897**	**0.8830**

Table 13. Encoding complexity comparison at resolution 1920×1080 under video regression. The encoding time is evaluated by an NVIDIA V100 GPU.

Method	Size	Encoding time	UVG PSNR
NeRV [7]	3.04M	1h37m27s	31.11
NeRV-Boost	3.06M	3h54m15s	32.76
E-NeRV [23]	3.01M	2h47m42s	31.51
E-NeRV-Boost	3.03M	5h14m23s	33.40
HNeRV [9]	3.05M	7h3m34s	32.47
HNeRV-Boost	3.06M	13h51m0s	33.89

Table 14. Additional implicit models in video regression on the UVG dataset.

Model	Size	Resolution	PSNR Avg.	MS-SSIM Avg.
DNeRV [15]	8.02M	1024×1920	34.73	0.9498
DNeRV-Boost	8.08M	1024×1920	**35.50**	**0.9548**
DivNeRV [55]	10.08M	1080×1920	33.93	0.9316
DivNeRV-Boost	10.02M	1080×1920	**35.35**	**0.9521**

Table 15. Ablation study of different boosting components on the Buuny video with 3M model size and 300 training epochs.

Model	TAT	SNeRV	Improved loss	PSNR
NeRV [7]				31.84
	✓			33.50
	✓	✓		36.28
	✓	✓	✓	**37.25**
E-NeRV [23]				37.32
	✓			38.01
	✓	✓		39.34
	✓	✓	✓	**40.07**
HNeRV [9]				38.15
	✓			39.90
	✓	✓		40.19
	✓	✓	✓	**41.09**

Table 16. Video inpainting results on the DAVIS validation dataset in MS-SSIM.

Video	Mask-S						Mask-C					
	NeRV	NeRV-Boost	E-NeRV	E-NeRV-Boost	HNeRV	HNeRV-Boost	NeRV	NeRV-Boost	E-NeRV	E-NeRV-Boost	HNeRV	HNeRV-Boost
Blackswan	0.8687	**0.9234**	0.9216	**0.9362**	0.9002	**0.9626**	0.8314	**0.8869**	0.8857	**0.9021**	0.8736	**0.9292**
Bmx-trees	0.8421	**0.9124**	0.8905	**0.9239**	0.8676	**0.9519**	0.8091	**0.8780**	0.8537	**0.8992**	0.8308	**0.9062**
Breakdance	0.9347	**0.9497**	0.9571	**0.9665**	0.9184	**0.9791**	0.8857	**0.9016**	0.9083	**0.9152**	0.9172	**0.9257**
Camel	0.8178	**0.8699**	0.8790	**0.9011**	0.8577	**0.9480**	0.7784	**0.8319**	0.8397	**0.8689**	0.8186	**0.9009**
Car-roundabout	0.8741	**0.9301**	0.9151	**0.9407**	0.9194	**0.9596**	0.8323	**0.8868**	0.8752	**0.9047**	0.8798	**0.9149**
Car-shadow	0.9042	**0.9555**	0.9515	**0.9558**	0.9367	**0.9685**	0.8673	**0.9132**	0.9059	**0.9158**	0.7635	**0.9256**
Cows	0.7408	**0.8445**	0.8346	**0.9026**	0.7983	**0.9408**	0.6957	**0.8041**	0.7917	**0.8564**	0.7627	**0.9022**
Dance-twirl	0.8296	**0.8866**	0.8682	**0.8959**	0.8544	**0.9223**	0.7891	**0.8457**	0.8265	**0.8569**	0.8074	**0.8842**
Dog	0.8770	**0.9321**	0.9382	**0.9455**	0.8297	**0.9636**	0.8383	**0.8921**	0.8995	**0.9107**	0.8655	**0.9212**
Drift-chicane	0.9747	**0.9870**	0.9870	**0.9899**	0.9806	**0.9924**	0.9360	**0.9435**	0.9539	**0.9583**	0.9360	**0.9484**
Drift-straight	0.9134	**0.9612**	0.9499	**0.9670**	0.9261	**0.9820**	0.8662	**0.9160**	0.9040	**0.9260**	0.8452	**0.9324**
Goat	0.8116	**0.8661**	0.8585	**0.8885**	0.8699	**0.9471**	0.7709	**0.8239**	0.8187	**0.8462**	0.8270	**0.8972**
Horsejump-high	0.8861	**0.9306**	0.9238	**0.9316**	0.9079	**0.9416**	0.8462	**0.8905**	0.8847	**0.8937**	0.8641	**0.8997**
Kite-surf	0.9048	**0.9511**	0.9504	**0.9592**	0.9226	**0.9725**	0.8676	**0.9136**	0.9146	**0.9258**	0.8781	**0.9302**
Libby	0.8895	**0.9504**	0.9355	**0.9576**	0.7921	**0.9756**	0.8539	**0.9123**	0.8956	**0.9232**	0.7373	**0.9372**
Motocross-jump	0.9251	**0.9702**	0.9601	**0.9683**	0.8524	**0.9747**	0.8930	**0.9360**	0.9271	**0.9314**	0.8319	**0.9403**
Paragliding-launch	0.8615	**0.9063**	0.8999	**0.9214**	0.8920	**0.9416**	0.8366	**0.8800**	0.8769	**0.8970**	0.8626	**0.9158**
Parkour	0.8152	**0.8840**	0.8565	**0.8888**	0.8399	**0.9142**	0.7780	**0.8493**	0.8203	**0.8554**	0.7998	**0.8706**
Scooter-black	0.8894	**0.9422**	0.9406	**0.9521**	0.9349	**0.9647**	0.8480	**0.8960**	0.8925	**0.9058**	0.8849	**0.9145**
Soapbox	0.8689	**0.9159**	0.8960	**0.9231**	0.8907	**0.9392**	0.8334	**0.8801**	0.8570	**0.8877**	0.8483	**0.8917**
Average	0.8715	**0.9235**	0.9157	**0.9358**	0.8846	**0.9571**	0.8329	**0.8841**	0.8766	**0.8990**	0.8417	**0.9144**

Table 17. Details of each module. Conv(input channels, output channels, kernel size, stride). PixelShuffle(upscale factor). r denotes the reduction factor of the channel width.

TAT(C)		TAT Resblock(C)	SNeRV(C, r, k, s)	Sinusoidal E-NeRV(C, r, k, s)
Conv(32, 32, 1, 1)	Conv(32, 32, 1, 1)	TAT(C)	Conv(C, $C * s * s/r$, k, 1)	Conv(C, $C * s * s/4$, k, 1)
ReLU()	ReLU()	Conv(C, C, 3, 1)	$\begin{cases} \text{Identity}(), s = 1 \\ \text{PixelShuffle}(s), s > 1 \end{cases}$	PixelShuffle(s)
Conv(32, C, 1, 1)	Conv(32, C, 1, 1)	GELU()		Conv($C/4$, C/r, 3, 1)
		TAT(C)		SINE()
		Conv(C, C, 3, 1)	SINE()	
		Skip connection		

Table 18. Details of conditional decoders in different implicit models. $C_{i+1} = \lfloor C_i/r, C_{min} \rfloor$, where the minimum channel width C_{min} is set as 12.

z_t generation network	NeRV-Boost	E-NeRV-Boost	HNeRV-Boost
Reshape	SNeRV(C_1, 1, 3, s_1)	Sinusoidal E-NeRV(C_1, 1/3, 3, s_1)	SNeRV(16, 16/C_1, 1, 1)
Conv(160, 64, 1, 1)	TAT(C_1)	TAT($C_1 * 3$)	TAT(C_1)
SINE()	SNeRV(C_1, 2, 3, s_2)	SNeRV($C_1 * 3$, 2, 3, s_2)	SNeRV(C_1, 1.2, 3, s_1)
Conv(64, 32, 1, 1)	TAT(C_2)	TAT(C_2)	TAT(C_2)
SINE()	SNeRV(C_2, 2, 3, s_3)	SNeRV(C_2, 2, 3, s_3)	SNeRV(C_2, 1.2, 3, s_2)
	TAT(C_3)	TAT(C_3)	TAT(C_3)
	SNeRV(C_3, 1, 3, 1)	SNeRV(C_3, 1, 3, 1)	SNeRV(C_3, 1.2, 3, s_3)
	TAT(C_3)	TAT(C_3)	TAT(C_4)
	SNeRV(C_3, 2, 3, s_4)	SNeRV(C_3, 2, 3, s_4)	SNeRV(C_4, 1, 3, 1)
	TAT(C_4)	TAT(C_4)	TAT(C_4)
	SNeRV(C_4, 1, 3, 1)	SNeRV(C_4, 1, 3, 1)	SNeRV(C_4, 1.2, 3, s_4)
	TAT(C_4)	TAT(C_4)	TAT(C_5)
	SNeRV(C_4, 2, 3, s_5)	SNeRV(C_4, 2, 3, s_5)	SNeRV(C_5, 1, 3, 1)
	TAT(C_5)	TAT(C_5)	TAT(C_5)
	SNeRV(C_5, 1, 3, 1)	SNeRV(C_5, 1, 3, 1)	SNeRV(C_5, 1.2, 3, s_5)
	TAT(C_5)	TAT(C_5)	TAT(C_6)
	Conv(C_5, 3, 1, 1)	Conv(C_5, 3, 1, 1)	SNeRV(C_6, 1, 3, 1)
			TAT(C_6)
			Conv(C_6, 3, 1, 1)

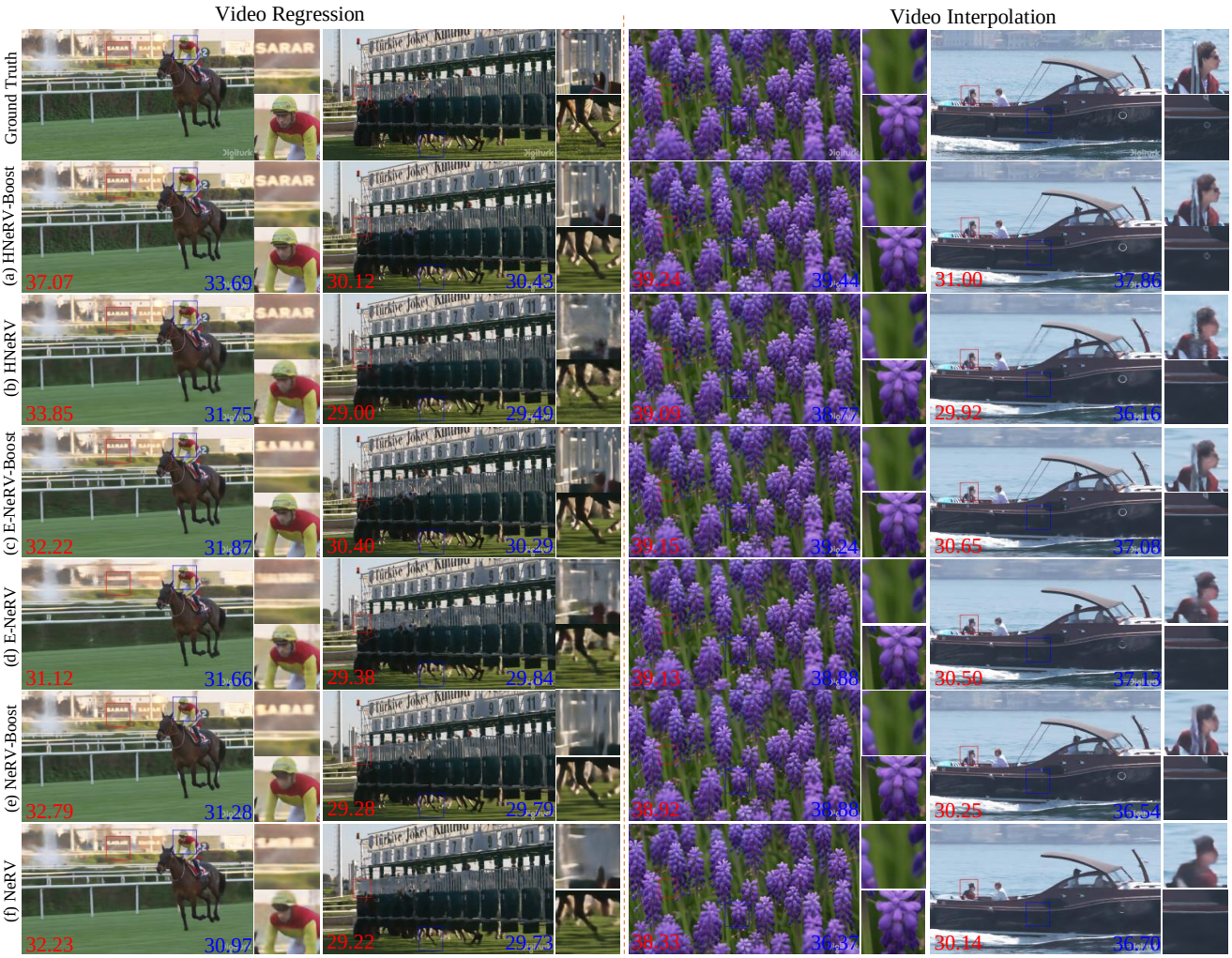

Figure 9. Visualization comparison of regression and interpolation on the UVG dataset. The top row displays the ground truth, followed by our boosted results in (a, c, e) and the baseline results in (b, d, f). The red and blue numbers indicate the PSNR values for the respective colored patches.

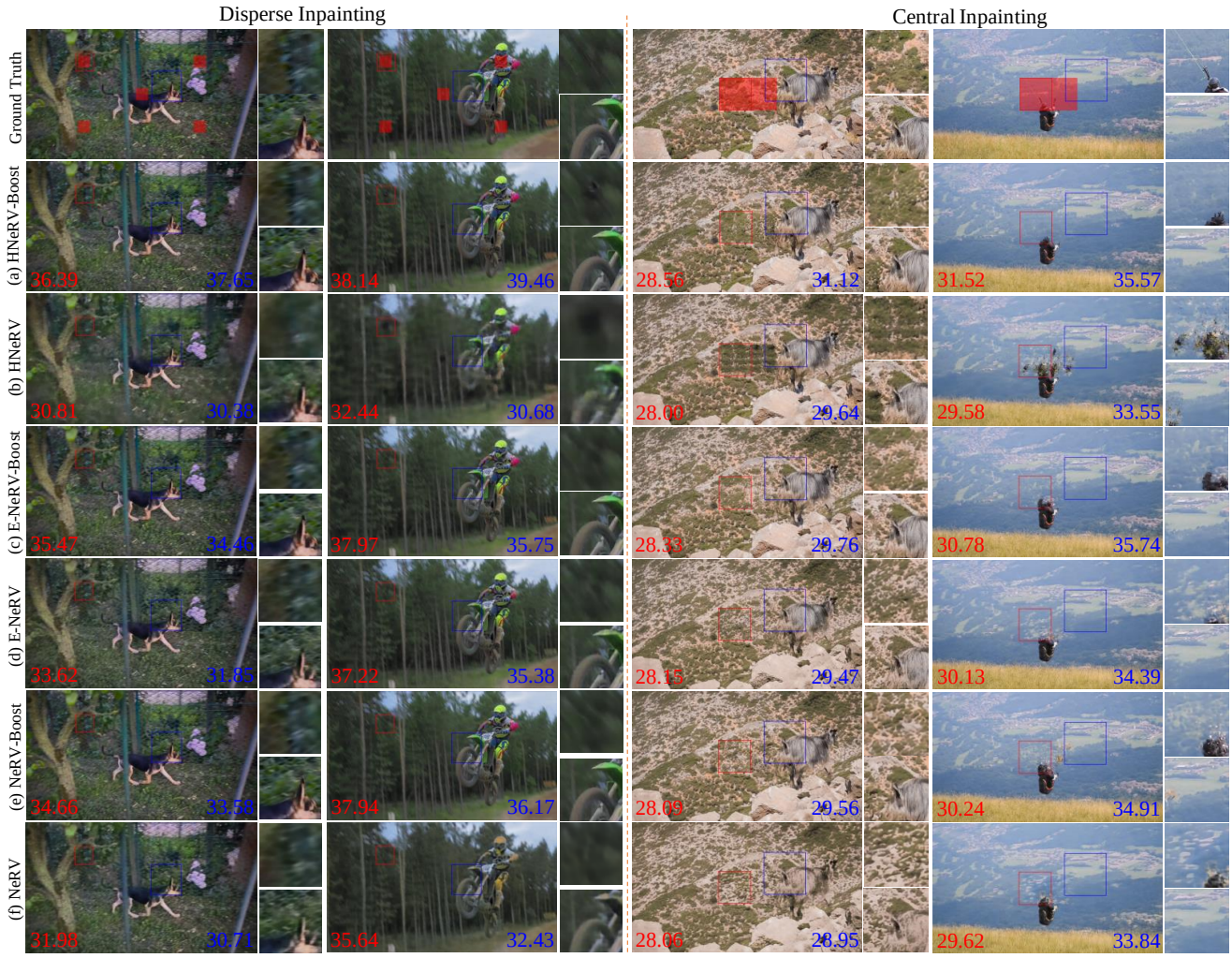

Figure 10. Visualization comparison of disperse and central inpainting on the DAVIS validation dataset. The top row displays the ground truth, followed by our boosted results in (a, c, e) and the baseline results in (b, d, f). The red and blue numbers indicate the PSNR values for the respective colored patches.

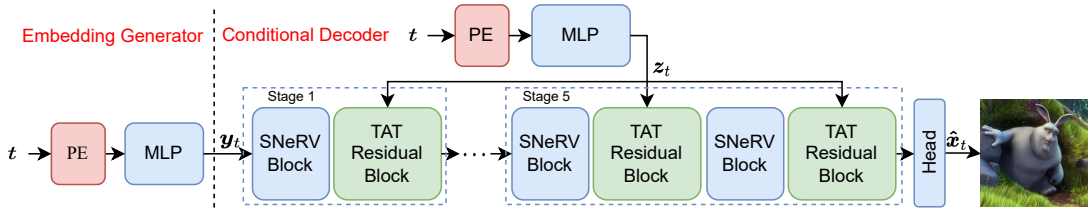

Figure 11. Our proposed NeRV-Boost framework with the conditional decoder. We follow the original NeRV to upsample the embedding $\boldsymbol{y}_t$ in stages 1 to 5.

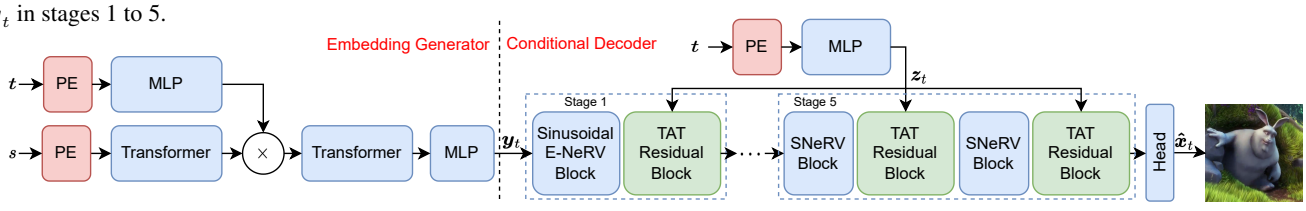

Figure 12. Our proposed E-NeRV-Boost framework with the conditional decoder. We follow the original E-NeRV to adopt the E-NeRV block in the stage 1, while the GELU activation is replaced with the SINE activation in the E-NeRV block. The embedding $\boldsymbol{y}_t$ is progressively upsampled and modulated in stages 1 to 5.